CS6923 Machine Learning, Fall 2018
Prof. Hellerstein, NYU School of Engineering

Homework 3

Submit on NYU Classes by Mon. Nov. 5 at 6:00 p.m. You may work together with one other person on this homework. If you do that, hand in JUST ONE homework for the two of you, with both of your names on it. You may *discuss* this homework with other students but YOU MAY NOT SHARE WRITTEN ANSWERS OR CODE WITH ANYONE BUT YOUR PARTNER.

IMPORTANT SUBMISSION INSTRUCTIONS: Please submit your solutions in 3 separate files: one file for your written answers to Part I, one file for your written answers/output for the questions in Part II, and one file with your code (a zip file if your code requires more than one file).

Part I: Written Exercises

1. The entropy of a dataset S is defined as follows:

$$Entropy(S) = -\sum_{l \in L} \frac{N_l}{N} \log_2 \frac{N_l}{N}$$

where L is the set of class labels for the examples in S (e.g., + or -), N is the number of examples in S, and N_l is the number of examples in S that have label l.

(In lecture, we focused on the definition for the case where there are 2 classes. The above definition is the generalization to k class problems, for $k \geq 2$. Note that the logarithm is base 2, even with more than 2 classes! It is standard when defining the entropy of a random variable to use the same base for the logarithm, no matter the size of the domain in question. We use base 2, which is one standard choice. The other standard choice is to take the natural log.)

Let x_i be a discrete-valued attribute of the examples in dataset S, and let V be the set of possible values of attribute x_i. For $v \in V$, let S_v be the subset of examples in S satisfying $x_i = v$.

Consider a classification problem with categorical attributes. Applied to such a problem, the decision tree algorithm discussed in class builds a decision tree where each node is labeled by an attribute x_i. When building the tree, at each node, the algorithm chooses the decision (i.e., the x_i) that minimizes

$$\sum_{v \in V} \frac{|S_v|}{|S|} Entropy(S_v)$$

This is equivalent to saying that it chooses the x_i that maximizes the following quantity, which is sometimes called the Information Gain of x_i (with respect to S)

$$\text{Information-Gain}(S) = \mathit{Entropy}(S) - \sum_{v \in V} \frac{|S_v|}{|S|} \mathit{Entropy}(S_v)$$

It can be shown that Information-Gain(S) is always greater than or equal to 0. Recall that we consider $0 \log_2 0$ to be equal to 0.

(a) Answer the following question without using a calculator. Which dataset has higher entropy: A dataset with 4 positive examples and 5 negative examples, or a dataset with 3 positive examples and 6 negative examples?

(b) Calculate the Information Gain of x_1 in the following dataset. Show your work. Two of the examples have identical attribute values. You should treat them as 2 separate examples in all of your calculations.

	x_1	x_2	x_3	r
$x^{(1)}$	F	F	F	+
$x^{(2)}$	T	F	T	+
$x^{(3)}$	F	F	T	-
$x^{(4)}$	F	T	F	+
$x^{(5)}$	T	T	F	-
$x^{(6)}$	T	T	T	-
$x^{(7)}$	F	F	F	-

(Note: Sometimes datasets will have 2 examples with the same attribute values, but different labels. This is sometimes due to an error in the labeling, but can also just reflect the fact that for a given vector of attribute values x, there isn't only one correct label. Consider a dataset where each example corresponds to a customer, and the label indicates whether the customer bought a certain product. One customer might buy the product, and another customer with the same attribute values might not.

As we described it in class, a decision tree algorithm with post-pruning will first grow the tree until it has 0% training error, and then prune it to be smaller. But if two examples have the same attribute values but with different labels, this isn't possible.)

(c) Run the decision tree algorithm described in class (without any pruning) to produce a tree on the dataset in part (b) having the smallest error possible. You will need to modify the algorithm from class slightly, since it is not possible to achieve 0% error (due to the two examples with the same attribute values but different labels). Form a leaf of the tree whenever all examples reaching the leaf have the same label, OR all examples reaching the leaf have the same values

for all the attributes. For any leaf containing examples that all have the same attribute values, predict the majority label, or + if there is a tie.

Note: There are better ways to handle having examples with the same attribute values but different labels! Usually you would do at least some limited pre-pruning, and form a leaf when a node either has few examples reaching it, or is almost pure (even if you'd then apply post-pruning). We're keeping it simple here for the homework exercise.

	x_1	x_2	x_3	r
$x^{(1)}$	F	F	F	+
$x^{(2)}$	T	F	T	+
$x^{(3)}$	F	F	T	-
$x^{(4)}$	F	T	F	+
$x^{(5)}$	T	T	F	-
$x^{(6)}$	T	T	T	-
$x^{(7)}$	F	F	F	+

(d) In information theory, the entropy of a discrete random variable taking values in $\{1, \ldots, n\}$ is $-\sum_{i=1}^{n} P[X = i] \log_2 P[X = i]$. This is the expected number of bits needed to encode a randomly drawn value of X, under the "most efficient"code. It is standard to use $H(X)$ to denote the entropy of X.

Given two discrete random variable, X and Y, both taking values in a finite set, the conditional entropy of Y given X, written $H(Y|X)$ is defined to be

$$H(Y|X) = \sum_x P[X = x] * \left(\sum_y -P[Y = y|X = x] * \log_2 P[Y = y|X = x] \right)$$

Here $\sum_x$ and $\sum_y$ denote, respectively, the sum over all possible values x of X and y of Y. The quantity

$$H(Y) - H(Y|X)$$

is called the *mutual information* between X and Y. Interestingly, this quantity is symmetric with respect to X and Y, and is also equal to

$$H(X) - H(X|Y)$$

We can relate this quantity to our definition of Information Gain. Consider an example drawn uniformly at random from from dataset S. Let Y be the label of the example, and let X be the value of attribute x_i in the example. Using the definition of $H(Y|X)$, we see that the information gain of x_i in dataset S is $H(Y) - H(Y|X)$. This

is why the Information Gain of x_i is sometimes called the mutual information betwen attribute x_i and the label.

Consider the dataset in part (b) again. Let Y be the label of an example drawn uniformly at random from the dataset. let X be the value of x_1 in the example. What are the values of $H(X)$ and $H(Y|X)$? What is the value of $H(X) - H(Y|X)$?

(e) What is the entropy of a labeled dataset S, with z possible labels, if each label appears equally often among the examples in the dataset?

Part II: Programming Exercises

2. This problem will give you practice with some of the linear algebra involved in PCA. For a quick review the basics of eigenvalues and eigenvectors, see the following short webpage: `http://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/eigen/eigen.html`.

   (a) By hand, find the characteristic polynomial of the matrix $A =$

   $$\begin{bmatrix} 0 & 14 \\ 6 & 9 \end{bmatrix}$$

   (b) By hand, solve for the eigenvalues of A using the characteristic polynomial you just computed.

   (c) Solve for the eigenvectors of A using those eigenvalues. The L_2 norm of your eigenvectors should be equal to 1.

   (d) Validate your results in Python (numpy) using linalg.eig(A) using the command [V,D]=eig(A).

   Give the returned eigenvalues and eigenvectors.

3. (a) Consider the following training set, consisting of 4 unlabeled examples, with real-valued attributes x_1, x_2, and x_3. The training set is represented by a matrix X, where each row of the matrix corresponds to an example, and column i corresponds to the ith attribute.

   $$\begin{bmatrix} 5 & 2 & 4 \\ 9 & 6 & 4 \\ 7 & 1 & 0 \\ 2 & 5 & 6 \end{bmatrix}$$

   Begin by centering the data, causing the sample mean in each dimension to be 0, as follows. Calculate the sample means for each column $[s_1, s_2, s_3]$ by hand, and then subtract these values from the entries in each column (e.g. s_3 is subtracted from all values in column 3). Call the resulting training set (matrix) B. Show the matrix B.

(b) By hand, calculate $s_{1,3}$, the sample covariance of x_1 and x_3 in B. Show your result.

Recall that the sample covariance of x_i and x_j for dataset a dataset with N examples is

$$\frac{\sum_t (x_i^t - s_i) * (x_j^t - s_j)}{N - 1}$$

(c) For the same B as in the previous question, use the Python command np.cov(B) to produce the sample covariance matrix of X. Entry $1, 2$ should be equal to the value you computed in the previous question (ignoring round-off errors).

Sample covariance matrices are symmetric positive semi-definite. (By definition, a symmetric matrix is positive semi-definite if all of its eigenvalues are non-negative.)

What is the largest eigenvalue of the sample covariance matrix of B? Use Python to compute this answer.

(d) Again using the same B as in the previous question, run:

```
import sklearn.decomposition as skd
import numpy as np
# .fit computes the principal components  (n_components of them)
# The columns of W are the eigenvectors of the covariance matrix of X
pca = skd.PCA(n_components = 3)
skd.PCA.fit(pca,X)
W1 = pca.components_
W = W1.transpose()
Z = pca.transform(X)
```

The first column of the matrix W is a vector w of norm 1, such that the projection of the data in X in the direction of w has the largest variance possible. That is, it is the solution to the problem of maximizing the variance of the entries of w'X' subject to the constraint $||w|| = 1$.

Z is a new representation of the data in matrix X, using the attributes produced by PCA, rather than the original attributes. The first column of Z corresponds to the attribute associated with the principal (largest) eigenvalue of the sample covariance matrix. The second column of Z corresponds to the attribute associated with the second largest eigenvalue of the sample covariance matrix, etc.

If we wanted to do dimension reduction on X, and reduce the dimension of the data to 2 (using new, transformed features), we could replace X by the first 2 columns of Z.

Show the first two columns of Z.

5. Consider the Python commands above.

PCA does not just project the data, it also centers it around the origin (which doesn't affect the variance). This is accomplished by computing the sample mean m of the examples (rows) in X, and subtracting it from each row of X, prior to computing the projection. If x is the first row (example) of X, and z is the corresponding first row of z, then its first entry, z_1, is computed as follows: $z_1 = w^T * (x^T - m^T)$ or equivalently, $z_1 = (x - m) * w$.

More generally, if N is the number of examples (rows) in X, and M is the $N \times N$ matrix whose rows are all equal to m, we get $Z = (X - M) * W$.

Because the columns of W are orthogonal, and have unit length, $X = ZW^T + M$.

If we use PCA to reduce the dimension to k attributes, we want to use only the first k principal components, corresponding to the first k columns of W and Z. Let W_k and Z_k denote the matrices consisting of the first k columns of W and Z. In this case, we can use the same calculation, $X = ZW^T + M$, but with W_k and Z_k, to get an approximation to X.

Follow the instructions below and then answer the questions.

PYTHON

The code below loads a dataset containing images of faces and displays some of them. You will need to run the code to answer the given questions. (You may need to install pillow or something similar.)

Each image is a 62x47 pixel array. The images are read into a matrix called fea. The rows of the matrix fea are the images (examples). The features (columns) are the pixels. Each example is represented by a vector of real numbers of length 2914, listing the pixels from left to right, row by row, from top to bottom.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import fetch_lfw_people
lfw_people = fetch_lfw_people(min_faces_per_person=70)
n_samples, h, w = lfw_people.images.shape
npix = h*w
fea = lfw_people.data

def plt_face(x):
    global h,w
    plt.imshow(x.reshape((h, w)), cmap=plt.cm.gray)
    plt.xticks([])

plt.figure(figsize=(10,20))
nplt = 4
```

```
for i in range(nplt):
    plt.subplot(1,nplt,i+1)
    plt_face(fea[i])

plt.show()
```

You can display the face corresponding to Example t in the dataset by executing the following commands:

```
plt_face(fea[t])
plt.show()
```

(a) Display the fourth face in the dataset using the appropriate command above. Print the resulting display.

(b) Compute the mean of all the examples in the dataset fea. (That is, compute an image such that each pixel i of the image is the mean of pixel i in all the images in fea.) Display it using a modification of the above command. Give the Python commands you used and show the resulting display.

(c) Let's do dimensionality reduction with pca. Using Python, compute the 5 top principal components of the data matrix fea. Give the Python commands you used. What are the values of the associated 5 attributes of the fourth image in the dataset?

(d) Using the reconstruction equation $X = W^T Z + M$ described above, but with just the first 5 columns of Z and W (the attributes associated with the first 5 principal components), approximate the fourth image in the dataset. Give the Python commands you used. Display the resulting approximate image and print the resulting display. Repeat with the first 50 columns of Z and W.

(These are representations of the fourth image, based on 5 or 50 features instead of the original 2914 pixel features.)